Fast sparse matrix multiplication

Raphael Yuster [*] *Uri Zwick* [†]

Abstract

Let A and B two $n \times n$ matrices over a ring R (e.g., the reals or the integers) each containing at most m non-zero elements. We present a new algorithm that multiplies A and B using $O(m^{0.7}n^{1.2} + n^{2+o(1)})$ algebraic operations (i.e., multiplications, additions and subtractions) over R. The naive matrix multiplication algorithm, on the other hand, may need to perform $\Omega(mn)$ operations to accomplish the same task. For $m \leq n^{1.14}$, the new algorithm performs an almost optimal number of only $n^{2+o(1)}$ operations! For $m \leq n^{1.68}$, the new algorithm is also faster than the best known matrix multiplication algorithm for dense matrices which uses $O(n^{2.38})$ algebraic operations. Obtaining a fast sparse matrix multiplication algorithm that beats the naive method even for very sparse matrices was a long standing open problem. The new algorithm is obtained using a surprisingly straightforward combination of a simple combinatorial idea and existing fast *rectangular* matrix multiplication algorithms. We also obtain improved algorithms for the multiplication of more than two sparse matrices. As the known fast rectangular matrix multiplication algorithms are far from being practical, our result, at least for now, is only of theoretical value.

1 Introduction

The multiplication of two $n \times n$ matrices is one of the most basic algebraic problems and considerable effort was devoted to obtaining efficient algorithms for the task. The naive matrix multiplication algorithm performs $O(n^3)$ operations. Strassen [Str69] was the first to show that the naive algorithm is not optimal, giving an $O(n^{2.81})$ algorithm for the problem. Many improvements then followed. The currently fastest matrix multiplication algorithm, with a complexity of $O(n^{2.38})$, was obtained by Coppersmith and Winograd [CW90]. More information on the fascinating subject of matrix multiplication algorithms and its history can be found in Pan [Pan85] and Bürgisser *et al.* [BCS97]. An interesting new group theoretic approach to the matrix multiplication problem was recently suggested by Cohn and Umans [CU03]. For the best available lower bounds see Shpilka [Shp03] and Raz [Raz03].

Matrix multiplication has numerous applications in combinatorial optimization in general, and in graph algorithms in particular. Fast matrix multiplication algorithms can be used, for example, to obtain fast algorithms for finding simple cycles in graphs [AYZ95, AYZ97, YZ04], for finding small cliques and other small subgraphs [NP85], for finding shortest paths [Sei95, SZ99, Zwi02], for obtaining improved dynamic reachability algorithms [DI00, RZ02], and for matching problems [MVV87, RV89, Che97]. Other applications can be found in [Cha02, KS03], and this list is not exhaustive.

In many cases, the matrices to be multiplied are *sparse*, i.e., the number of non-zero elements in them is negligible compared to the number of zeros in them. For example if $G = (V, E)$ is a directed graph on n vertices containing m edges, then its adjacency matrix A_G is an $n \times n$ matrix with only m non-zero elements (1's in this case). In many interesting cases $m = o(n^2)$. Unfortunately, the fast matrix multiplication algorithms

[*]Department of Mathematics, University of Haifa at Oranim, Tivon 36006, Israel. E–mail: raphy@research.haifa.ac.il
[†]Department of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E–mail: zwick@post.tau.ac.il

mentioned above cannot utilize the sparsity of the matrices multiplied. The complexity of the algorithm of Coppersmith and Winograd [CW90], for example, remains $O(n^{2.38})$ even if the multiplied matrices are extremely sparse. The naive matrix multiplication algorithm, on the other hand, can be used to multiply two $n \times n$ matrices, each with at most m non-zero elements, using $O(mn)$ operations (see next section). Thus, for $m = O(n^{1.37})$, the sophisticated matrix multiplication algorithms of Coppersmith and Winograd [CW90] and others do not provide any improvement over the naive matrix multiplication algorithm.

In this paper we show that the sophisticated algebraic techniques used by the fast matrix multiplication algorithms *can* nevertheless be used to speed-up the computation of the product of even extremely sparse matrices. More specifically, we present a new algorithm that multiplies two $n \times n$ matrices, each with at most m non-zero elements, using $O(m^{0.7}n^{1.2} + n^{2+o(1)})$ algebraic operations. (The exponents 0.7 and 1.2 are derived, of course, from the current 2.38 bound on the exponent of matrix multiplication, and from bounds on other exponents related to matrix multiplications, as will be explained in the sequel.) There are three important things to notice here:

(*i*) If $m \geq n^{1+\epsilon}$, for any $\epsilon > 0$, then the number of operations performed by the new algorithm is $o(mn)$, i.e., less then the number of operations performed, in the worst-case, by the naive algorithm.

(*ii*) If $m \leq n^{1.14}$, then the new algorithm performs only $n^{2+o(1)}$ operations. This is very close to optimal as all n^2 entries in the product may be non-zero, even if the multiplied matrices are very sparse.

(*iii*) If $m \leq n^{1.68}$, then the new algorithm performs only $o(n^{2.38})$, i.e., fewer operations than the fastest known matrix multiplication algorithm.

In other words, the new algorithm improves on the naive algorithm even for extremely sparse matrices (i.e., $m = n^{1+\epsilon}$), and it improves on the fastest matrix multiplication algorithm even for relatively dense matrices (i.e., $m = n^{1.68}$).

The new algorithm is obtained using a surprisingly straightforward combination of a simple combinatorial idea, implicit in Eisenbrand and Grandoni [EG03] and Yuster and Zwick [YZ04], with the fast matrix multiplication algorithm of Coppersmith and Winograd [CW90], and the fast *rectangular* matrix multiplication algorithm of Coppersmith [Cop97]. It is interesting to note that a fast rectangular matrix multiplication algorithm for dense matrices is used to obtain a fast matrix multiplication algorithm for sparse square matrices.

As mentioned above, matrix multiplication algorithms are used to obtain fast algorithms for many different graph problems. We note (with some regret ...) that our improved sparse matrix multiplication algorithm does not yield, automatically, improved algorithms for these problems on sparse graphs. These algorithms may need to multiply dense matrices even if the input graph is sparse. Consider for example the computation of the transitive closure of a graph by repeatedly squaring its adjacency matrix. The matrix obtain after the first squaring may already be extremely dense. Still, we expect to find many situations in which the new algorithm presented here could be useful.

In view of the above remark, we also consider the problem of computing the product $A_1A_2 \cdots A_k$ of three or more sparse matrices. As the product of even very sparse matrices can be completely dense, the new algorithm for multiplying two matrices cannot be applied directly in this case. We show, however, that some improved bounds may also be obtained in this case. Our results here are less impressive, however. For $k = 3$, we improve, for certain densities, on the performance of all existing algorithms. For $k \geq 4$ we get no worst-case improvements at the moment, but such improvements will be obtained if bounds on certain matrix multiplication exponents are sufficiently improved.

The problem of computing the product $A_1A_2 \cdots A_k$ of $k \geq 3$ rectangular matrices, known as the *chain matrix multiplication problem*, was, of course, addressed before. The main concern, however, was finding an optimal

way of parenthesizing the expression so that a minimal number of operations will be performed when the naive algorithm is used to successively multiply pairs of intermediate results. Such an optimal placement of parentheses can be easily found in $O(k^3)$ time using dynamic programming (see, e.g., Chapter 15 of [CLRS01]). A much more complicated algorithm of Hu and Shing [HS82, HS84] can do the same in $O(k \log k)$ time. An almost optimal solution can be found in $O(k)$ time using a simple heuristic suggested by Chin [Chi78]. It is easy to modify the simple dynamic programming solution to the case in which fast rectangular matrix multiplication algorithm is used instead of the naive matrix multiplication algorithm. It is not clear whether the techniques of Hu and Shing and of Chin can also be modified accordingly. Cohen [Coh99] suggests an interesting technique for predicting the non-zero structure of a product of two or more matrices. Using her technique it is possible to exploit the possible sparseness of the intermediate products.

All these techniques, however, reduce the computation of a product like $A_1A_2A_3$ into the computation of A_1A_2 and then $(A_1A_2)A_3$, or to the computation of A_2A_3 and then $A_1(A_2A_3)$. We show that, for certain densities, a faster way exists.

The rest of the paper is organized as follows. In the next section we review the existing matrix multiplication algorithms. In Section 3 we present the main result of this paper, i.e., the improved sparse matrix multiplication algorithm. In Section 4 we use similar ideas to obtain an improved algorithm for the multiplication of three or more sparse matrices. We end, in Section 5, with some concluding remarks and open problems.

2 Existing matrix multiplication algorithms

In this short section we examine the worst-case behavior of the naive matrix multiplication algorithm and state the performance of existing fast matrix multiplication algorithms.

2.1 The naive matrix multiplication algorithm

Let A and B be two $n \times n$ matrices. The product $C = AB$ is defined as follows $c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}$, for $1 \leq i, j \leq n$. The naive matrix multiplication algorithm uses this definition to compute the entries of C using n^3 multiplications and $n^3 - n^2$ additions. The number of operations can be reduced by avoiding the computation of products $a_{ik}b_{kj}$ for which $a_{ik} = 0$ or $b_{kj} = 0$. In general, if we let $\bar{a}_k$ be the number of non-zero elements in the k-th column of A, and $\bar{b}_k$ be the number of non-zero elements in the k-th row of B, then the number of multiplications that need to be performed is only $\sum_{k=1}^{n} \bar{a}_k\bar{b}_k$. The number of additions required is always bounded by the required number of multiplications. This simple sparse matrix multiplication algorithm may be considered folklore. It can also be found in Gustavson [Gus78].

If A contains at most m non-zero entries, then $\sum_{k=1}^{n} \bar{a}_k\bar{b}_k \leq (\sum_{k=1}^{n} \bar{a}_k)n \leq mn$. The same bound is obtained when B contains at most m non-zero entries. Can we get an improved bound on the worst-case number of products required when both A and B are sparse? Unfortunately, the answer is no. Assume that $m \geq n$ and consider the case $\bar{a}_i = \bar{b}_i = n$, if $i \leq m/n$, and $\bar{a}_i = \bar{b}_i = 0$, otherwise. (In other words, all non-zero elements of A and B are concentrated in the first m/n columns of A and the first m/n columns of B.) In this case $\sum_{k=1}^{n} \bar{a}_k\bar{b}_k = (m/n) \cdot n^2 = mn$. Thus, the naive algorithm may have to perform mn multiplications even if both matrices are sparse. It is instructive to note that the computation of AB in this worst-case example can be reduced to the computation of a much smaller rectangular product. This illustrates the main idea behind the new algorithm: When the naive algorithm has to perform many operations, rectangular matrix multiplication can be used to speed up the computation.

To do justice with the naive matrix multiplication algorithm we should note that in many cases that appear in practice the matrices to be multiplied have a special structure, and the number of operations required may be

much smaller than mn. For example, if the non-zero elements of A are evenly distributed among the columns of A, and the non-zero elements of B are evenly distributed among the rows of B, we have $\bar{a}_k = \bar{b}_k = m/n$, for $1 \leq k \leq n$, and $\sum_{k=1}^{n} \bar{a}_k \bar{b}_k = n \cdot (m/n)^2 = m^2/n$. We are interested here, however, in worst-case bounds that hold for any placement of non-zero elements in the input matrices.

2.2 Fast matrix multiplication algorithms for dense matrices

Let $M(a, b, c)$ be the minimal number of algebraic operations needed to multiply an $a \times b$ matrix by a $b \times c$ matrix over an arbitrary ring R. Let $\omega(r, s, t)$ be the minimal exponent ω for which $M(n^r, n^s, n^t) = O(n^{\omega+o(1)})$. We are interested here mainly in $\omega = \omega(1, 1, 1)$, the exponent of square matrix multiplication, and $\omega(1, r, 1)$, the exponent of rectangular matrix multiplication of a particular form. The best bounds available on $\omega(1, r, 1)$, for $0 \leq r \leq 1$ are summarized in the following theorems:

Theorem 2.1 (Coppersmith and Winograd [CW90]) $\omega < 2.376$.

Next, we define two more constants, α and β, related to rectangular matrix multiplication.

Definition 2.2 $\alpha = \max\{\, 0 \leq r \leq 1 \mid \omega(1, r, 1) = 2 \,\}$, $\beta = \dfrac{\omega - 2}{1 - \alpha}$.

Theorem 2.3 (Coppersmith [Cop97]) $\alpha > 0.294$.

It is not difficult to see that these Theorems 2.1 and 2.3 imply the following theorem. A proof can be found, for example, in Huang and Pan [HP98].

Theorem 2.4 $\omega(1, r, 1) \leq \begin{cases} 2 & \text{if } 0 \leq r \leq \alpha, \\ 2 + \beta(r - \alpha) & \text{otherwise.} \end{cases}$

Corollary 2.5 $M(n, \ell, n) = n^{2-\alpha\beta+o(1)} \ell^{\beta} + n^{2+o(1)}$.

All the bounds in the rest of the paper will be expressed terms of α and β. Note that with $\omega = 2.376$ and $\alpha = 0.294$ we get $\beta \simeq 0.533$. If $\omega = 2$, as conjectured by many, then $\alpha = 1$. (In this case β is not defined, but also not needed.)

3 The new sparse matrix multiplication algorithm

Let A_{*k} be the k-th column of A, and let B_{k*} be the k-th row of B, for $1 \leq k \leq n$. Clearly $AB = \sum_k A_{*k} B_{k*}$. (Note that A_{*k} is a column vector, B_{k*} a row vector, and $A_{*k} B_{k*}$ is an $n \times n$ matrix.) Let a_k be the number of non-zero elements in A_{*k} and let b_k be the number of non-zero elements in B_{k*}. (For brevity, we omit the bars over a_k and b_k used in Section 2.1. No confusion will arise here.) As explained in Section 2.1, we can naively compute AB using $O(\sum_k a_k b_k)$ operations. If A and B each contain m non-zero elements, then $\sum_k a_k b_k$ may be as high as mn. (See the example in Section 2.1.)

For any subset $I \subseteq [n]$ let A_{*I} be the submatrix composed of the columns of A whose indices are in I and let B_{I*} the submatrix composed of the rows of B whose indices are in I. If $J = [n] - I$, then we clearly have $AB = A_{*I} B_{I*} + A_{*J} B_{J*}$. Note that $A_{*I} B_{I*}$ and $A_{*J} B_{J*}$ are both *rectangular* matrix multiplications.

Algorithm *SMP*(A, B)

Input: Two $n \times n$ matrices A and B.
Output: The product AB.

1. Let a_k be the number of non-zero elements in A_{*k}, the k-th column of A, for $1 \leq k \leq n$.
2. Let b_k be the number of non-zero elements in B_{k*}, the k-th row of B, for $1 \leq k \leq n$.
3. Let π be a permutation for which $a_{\pi(1)}b_{\pi(1)} \geq a_{\pi(2)}b_{\pi(2)} \geq \ldots \geq a_{\pi(n)}b_{\pi(n)}$.
4. Find an $0 \leq \ell \leq n$ that minimizes $M(n, \ell, n) + \sum_{k>\ell} a_{\pi(k)}b_{\pi(k)}$.
5. Let $I = \{\pi(1), \ldots, \pi(\ell)\}$ and $J = \{\pi(\ell+1), \ldots, \pi(n)\}$.
6. Compute $C_1 \leftarrow A_{*I}B_{I*}$ using the fast dense rectangular matrix multiplication.
7. Compute $C_2 \leftarrow A_{*J}B_{J*}$ using the naive sparse matrix multiplication algorithm.
8. Output $C_1 + C_2$.

Figure 1: The new fast sparse matrix multiplication algorithm.

Recall that $M(n, \ell, n)$ is the cost of multiplying an $n \times \ell$ matrix by an $\ell \times n$ matrix using the best available rectangular matrix multiplication algorithm.

Let π be a permutation for which $a_{\pi(1)}b_{\pi(1)} \geq a_{\pi(2)}b_{\pi(2)} \geq \ldots \geq a_{\pi(n)}b_{\pi(n)}$. A permutation π satisfying this requirement can be easily found in $O(n)$ time using radix sort. The algorithm chooses a value $1 \leq \ell \leq n$, in a way that will be specified shortly, and sets $I = \{\pi(1), \ldots, \pi(\ell)\}$ and $J = \{\pi(\ell+1), \ldots, \pi(n)\}$. The product $A_{*I}B_{I*}$ is then computed using the fastest available rectangular matrix multiplication algorithm, using $M(n, \ell, n)$ operations, while the product $A_{*J}B_{J*}$ is computed naively using $O(\sum_{k>\ell} a_{\pi(k)}b_{\pi(k)})$ operations. The two matrices $A_{*I}B_{I*}$ and $A_{*J}B_{J*}$ are added using $O(n^2)$ operations. We naturally choose the value ℓ that minimizes $M(n, \ell, n) + \sum_{k>\ell} a_{\pi(k)}b_{\pi(k)})$. (This can easily be done in $O(n)$ time by simply checking all possible values.) The resulting algorithm, which we call *SMP*, is given in Figure 1. We now claim:

Theorem 3.1 *Algorithm SMP*(A, B) *computes the product of two $n \times n$ matrices over a ring R, with m_1 and m_2 non-zero elements respectively, using at most*

$$O(\min\{\ (m_1 m_2)^{\frac{\beta}{\beta+1}}\, n^{\frac{2-\alpha\beta}{\beta+1}+o(1)} + n^{2+o(1)}\,,\ m_1 n\,,\ m_2 n\,,\ n^{\omega+o(1)}\ \})$$

ring operations.

If $m_1 = m_2 = m$, then the first term in the bound above becomes $m^{\frac{2\beta}{\beta+1}}\, n^{\frac{2-\alpha\beta}{\beta+1}+o(1)}$. It is easy to check that for $m = O(n^{1+\frac{\alpha}{2}})$, the number of operations performed by the algorithm is only $n^{2+o(1)}$. It is also not difficult to check that for $m = O(n^{\frac{\omega+1}{2}-\epsilon})$, for any $\epsilon > 0$, the algorithm performs only $o(n^\omega)$ operations. Using the currently best available bounds on ω, α and β, namely $\omega \simeq 2.376$, $\alpha \simeq 0.294$, and $\beta \simeq 0.536$, we get that the number of operations performed by the algorithm is at most $O(m^{0.7}n^{1.2} + n^{2+o(1)})$, justifying the claims made in the abstract and the introduction.

The proof of Theorem 3.1 relies on the following simple lemma:

Lemma 3.2 *For any $1 \leq \ell < n$ we have $\sum_{k>\ell} a_{\pi(k)}b_{\pi(k)} \leq \frac{m_1 m_2}{\ell}$.*

Proof: Assume, without loss of generality, that $a_1 \geq a_2 \geq \ldots \geq a_n$. Let $1 \leq \ell < n$. We then clearly have $\sum_{k>\ell} a_{\pi(k)} b_{\pi(k)} \leq \sum_{k>\ell} a_k b_k$. Also $\ell\, a_{\ell+1} \leq \sum_{k\leq\ell} a_k \leq m_1$. Thus $a_{\ell+1} \leq m_1/\ell$. Putting this together we get

$$\sum_{k>\ell} a_{\pi(k)} b_{\pi(k)} \;\leq\; \sum_{k>\ell} a_k b_k \;\leq\; a_{\ell+1} \sum_{k>\ell} b_k \;\leq\; \frac{m_1}{\ell}\, m_2 \;=\; \frac{m_1 m_2}{\ell}\,.$$

□

We are now ready for the proof of Theorem 3.1.

Proof: (of Theorem 3.1) We first examine the two extreme possible choices of ℓ. When $\ell = 0$, the naive matrix multiplication algorithm is used. When $\ell = n$, a fast dense square matrix multiplication algorithm is used. As the algorithm chooses the value of ℓ that minimized the cost, it is clear that the number of operations performed by the algorithm is $O(\min\{\, m_1 n\,,\, m_2 n\,,\, n^{\omega+o(1)}\,\})$.

All that remains, therefore, is to show that the number of operations performed by the algorithm is also $O((m_1 m_2)^{\frac{\beta}{\beta+1}}\, n^{\frac{2-\alpha\beta}{\beta+1}+o(1)} + n^{2+o(1)})$. If $m_1 m_2 \leq n^{2+\alpha}$, let $\ell = m_1 m_2/n^2$. As $\ell \leq n^\alpha$, we have,

$$M(n,\ell,n) + \sum_{k>\ell} a_{\pi(k)} b_{\pi(k)} \;\leq\; n^{2+o(1)} + \frac{m_1 m_2}{\ell} \;=\; n^{2+o(1)}\,.$$

If $m_1 m_2 \geq n^{2+\alpha}$, let $\ell = (m_1 m_2)^{\frac{1}{\beta+1}} n^{\frac{\alpha\beta-2}{\beta+1}}$. It is easy to verify that $\ell \geq n^\alpha$, and therefore

$$M(n,\ell,n) \;=\; n^{2-\alpha\beta+o(1)}\ell^\beta \;=\; (m_1 m_2)^{\frac{\beta}{\beta+1}} n^{\frac{2-\alpha\beta}{\beta+1}+o(1)}\,,$$

$$\sum_{k>\ell} a_{\pi(k)} b_{\pi(k)} \;\leq\; \frac{m_1 m_2}{\ell} \;=\; (m_1 m_2)^{1-\frac{1}{\beta+1}} n^{\frac{2-\alpha\beta}{\beta+1}} \;=\; (m_1 m_2)^{\frac{\beta}{\beta+1}} n^{\frac{2-\alpha\beta}{\beta+1}}\,.$$

Thus, $M(n,\ell,n) + \sum_{k>\ell} a_{\pi(k)} b_{\pi(k)} = (m_1 m_2)^{\frac{\beta}{\beta+1}} n^{\frac{2-\alpha\beta}{\beta+1}+o(1)}$. As the algorithm chooses the value of ℓ that minimizes the number of operations, this completes the proof of the theorem. □

4 Multiplying three or more matrices

In this section we extend the results of the previous section to the product of three or more matrices. Let $A_1, A_2, \ldots, A_k$ be $n \times n$ matrices, and let m_r, for $1 \leq r \leq k$ be the number of non-zero elements in A_r. Let $B = A_1 A_2 \cdots A_k$ be the product of the k matrices. As the product of two sparse matrices is not necessarily sparse, we cannot use the algorithm of the previous section directly to efficiently compute the product of more than two sparse matrices. Nevertheless, we show that the algorithm of the previous section can be generalized to efficiently handle the product of more than two matrices.

Let $A_r = (a^{(r)}_{ij})$, for $1 \leq r \leq k$, and $B = A_1 A_2 \cdots A_k = (b_{ij})$. It follows easily from the definition of matrix multiplication that

$$b_{ij} = \sum_{r_1, r_2, \ldots, r_{k-1}} a^{(1)}_{i,r_1} a^{(2)}_{r_1,r_2} \cdots a^{(k-1)}_{r_{k-2} r_{k-1}} a^{(k)}_{r_{k-1} j}\,. \tag{1}$$

It is convenient to interpret the computation of b_{ij} as the summation over paths in a layered graph, as shown (for the case $k = 4$) in Figure 2. More precisely, the layered graph corresponding to the product $A_1 A_2 \cdots A_k$ is composed of $k+1$ layers $V_1, V_2, \ldots, V_{k+1}$. Each layer V_r, where $1 \leq r \leq k+1$ is composed of n vertices $v_{r,i}$, for $1 \leq i \leq n$. For each non-zero element a^r_{ij} in A_r, there is an edge $v_{r,i} \rightarrow v_{r+1,j}$ in the graph labelled by the element a^r_{ij}. The element b_{ij} of the product is then the sum over all directed paths in the graph from $v_{1,i}$ to $v_{k+1,j}$ of the product of the elements labelling the edges of the path.

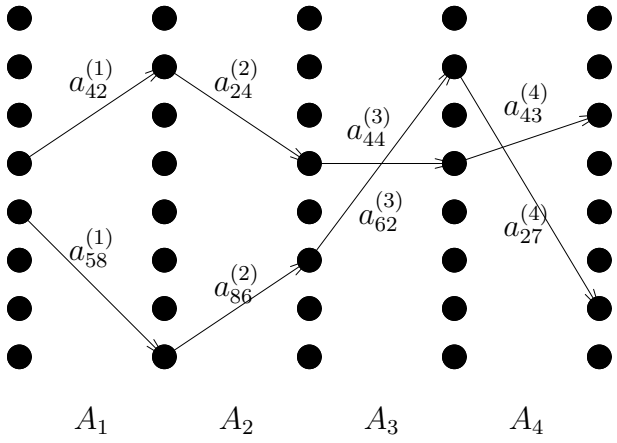

Figure 2: A layered graph corresponding to the product $A_1A_2A_3A_4$.

Algorithm *SCMP* given in Figure 3 is a generalization of the variant of algorithm *SMP* given in the previous section for the product of two matrices. The algorithm starts by setting ℓ to $(\prod_{r=1}^{k} m_r)^{\frac{1}{k-1+\beta}} n^{\frac{\alpha\beta-2}{k-1+\beta}}$, where m_r in the number of non-zero entries in A_r, for $1 \leq r \leq k$. (Note that when $k = 2$, we have $\ell = (m_1 m_2)^{\frac{1}{\beta+1}} n^{\frac{\alpha\beta-2}{\beta+1}}$, as in the proof of Theorem 3.1.) Next, the algorithm lets I_r be the set of indices of the ℓ rows of A_r with the largest number of non-zero elements, ties broken arbitrarily, for $2 \leq r \leq k$. It also lets $J_r = [n] - I_r$ be the set of indices of the $n - \ell$ rows of A_r with the smallest number of non-zero elements. The rows of A_r with indices in I_r are said to be the *heavy* rows of A_r, while the rows of A_r with indices in J_r are said to be *light* rows. The algorithm is then ready to do some calculations. For every $1 \leq r \leq k$, it computes $P_r \leftarrow (A_1)_{*J_2}(A_2)_{J_2 J_3} \cdots (A_r)_{J_r *}$. This is done by enumerating all the corresponding paths in the layered graph corresponding to the product. The matrix P_r is an $n \times n$ matrix that gives the contribution of the *light* paths, i.e., paths that do not use elements from *heavy* rows of $A_2, \ldots, A_r$, to the prefix product $A_1 A_2 \cdots A_r$. Next, the algorithm computes the suffix products $S_r \leftarrow A_r \cdots A_k$, for $2 \leq r \leq k$, using recursive calls to the algorithm. The cost of these recursive calls, as we shall see, will be overwhelmed by the other operations performed by the algorithm. The crucial step of the algorithm is the computation of $B_r \leftarrow (P_{r-1})_{*I_r}(S_r)_{I_r *}$, for $2 \leq r \leq k$, using the fastest available rectangular matrix multiplication algorithm. The algorithm then computes and outputs the matrix $(\sum_{r=2}^{k} B_r) + P_k$.

Theorem 4.1 *Let $A_1, A_2, \ldots, A_k$ be $n \times n$ matrices each with $m_1, m_2, \ldots, m_k$ non-zero elements, respectively, where $k \geq 2$ is a constant. Then, algorithm SCMP$(A_1, A_2, \ldots, A_k)$ correctly computes the product $A_1 A_2 \cdots A_k$ using*

$$O\left((\prod_{r=1}^{k} m_r)^{\frac{\beta}{k-1+\beta}} n^{\frac{(2-\alpha\beta)(k-1)}{k-1+\beta}+o(1)} + n^{2+o(1)}\right)$$

algebraic operations.

Proof: It is easy to see that the outdegree of a vertex $v_{r,j}$ is the number of non-zero elements in the j-th row of A_r. We say that a vertex $v_{r,j}$ is heavy if $j \in I_r$, and *light* otherwise. (Note that vertices of V_{k+1} are not classified as light or heavy. The classification of V_1 vertices is not used below.) A path in the layered graph is said to be light if all its *intermediate* vertices are light, and heavy if at least one of its *intermediate* vertices is heavy.

Let $a_{i,s_1}^{(1)} a_{s_1,s_2}^{(2)} \cdots a_{s_{k-2}s_{k-1}}^{(k-1)} a_{s_{k-1}j}^{(k)}$ be one of the terms appearing in the sum of b_{ij} given in (1). To prove the correctness of the algorithm we show that this term appears in exactly one of the matrices $B_2, \ldots, B_k$ and P_k

Algorithm $SCMP(A_1, A_2, \ldots, A_k)$

Input: $n \times n$ matrices $A_1, A_2, \ldots, A_k$.
Output: The product $A_1 A_2 \cdots A_k$.

1. Let $\ell = (\prod_{r=1}^{k} m_r)^{\frac{1}{k-1+\beta}} n^{\frac{\alpha\beta-2}{k-1+\beta}}$.
2. Let I_r be the set of indices of ℓ rows of A_r with the largest number of non-zero elements, and let $J_r = [n] - I_r$, for $2 \leq r \leq k$.
3. Compute $P_r \leftarrow (A_1)_{*J_2}(A_2)_{J_2 J_3} \cdots (A_r)_{J_r *}$ by enumerating all corresponding paths, for $1 \leq r \leq k$.
4. Compute $S_r \leftarrow A_r \cdots A_k$, for $2 \leq r \leq k$, using recursive calls to the algorithm.
5. Compute $B_r \leftarrow (P_{r-1})_{*I_r}(S_r)_{I_r *}$ using the fastest available rectangular matrix multiplication algorithm, for $2 \leq r \leq k$.
6. Output $(\sum_{r=2}^{k} B_r) + P_k$.

Figure 3: Computing the product of several sparse matrices.

which are added up to produce the matrix returned by the algorithm. Indeed, if the path corresponding to the term is light, then the term appears in P_k. Otherwise, let v_{r,j_r} be the first heavy vertex appearing on the path. The term then appears in B_r and in no other product. This completes the correctness proof.

We next consider the complexity of the algorithm. As mentioned, the outdegree of a vertex $v_{r,j}$ is equal to the number of non-zero elements in the j-th row of A_r. The total number of non-zero elements in A_r is m_r. Let d_r be the maximum outdegree of a light vertex of V_r. The outdegree of every heavy vertex of V_r is then at least d_r. As there are ℓ heavy vertices, it follows that $d_r \leq m_r/\ell$, for $2 \leq r \leq k$.

The most time consuming operations performed by the algorithm is the computation of

$$P_k \leftarrow (A_1)_{*J_2}(A_2)_{J_2 J_3} \cdots (A_k)_{J_k *}$$

by explicitly going over all light paths in the layered graph, and the $k-1$ rectangular products

$$B_r \leftarrow (P_{r-1})_{*I_r}(S_r)_{I_r *} \quad , \quad \text{for } 2 \leq r \leq k \; .$$

The number of light paths in the graph is at most $m_1 \cdot d_2 d_3 \cdots d_k$. Using the bounds we obtained on the d_r's, and the choice of ℓ we get that the number of light paths is at most

$$m_1 \cdot d_2 d_3 \cdots d_k \;\leq\; (\prod_{r=1}^{k} m_r)/\ell^{k-1}$$

$$\leq\; (\prod_{r=1}^{k} m_r) \left[(\prod_{r=1}^{k} m_r)^{-\frac{k-1}{k-1+\beta}} n^{\frac{(2-\alpha\beta)(k-1)}{k-1+\beta}} \right] \;=\; (\prod_{r=1}^{k} m_r)^{\frac{\beta}{k-1+\beta}} n^{\frac{(2-\alpha\beta)(k-1)}{k-1+\beta}} \; .$$

Thus, the time taken to compute P_k is $O((\prod_{r=1}^{k} m_r)^{\frac{\beta}{k-1+\beta}} n^{\frac{(2-\alpha\beta)(k-1)}{k-1+\beta}})$. (Computing the product of the elements along a path requires k operations, but we consider k to be a constant.)

As $|I_r| = \ell$, for $2 \leq r \leq k$, the product $(P_{r-1})_{*I_r}(S_r)_{I_r *}$ is the product of an $n \times \ell$ matrix by an $\ell \times n$ matrix whose cost is $M(n, \ell, n)$. Using Corollary 2.5 and the choice of ℓ made by the algorithm we get that

$$M(n, \ell, n) \;=\; n^{2-\alpha\beta+o(1)} \ell^{\beta} + n^{2+o(1)} \;=\; n^{2-\alpha\beta+o(1)} \left[(\prod_{r=1}^{k} m_r)^{\frac{\beta}{k-1+\beta}} n^{\frac{(\alpha\beta-2)\beta}{k-1+\beta}} \right] + n^{2+o(1)}$$

$$= \left(\prod_{r=1}^{k} m_r\right)^{\frac{\beta}{k-1+\beta}} n^{(2-\alpha\beta)(1-\frac{\beta}{k-1+\beta})+o(1)} + n^{2+o(1)} = \left(\prod_{r=1}^{k} m_r\right)^{\frac{\beta}{k-1+\beta}} n^{\frac{(2-\alpha\beta)(k-1)}{k-1+\beta}+o(1)} + n^{2+o(1)} .$$

Finally, it is easy to see that the cost of computing the suffix products $S_r \leftarrow A_r \cdots A_k$, for $2 \leq r \leq k$, using recursive calls to the algorithm, is dominated by the cost of the other operations performed by the algorithm. (Recall again that k is a constant.) This completes the proof of the theorem. □

There are two alternatives to the use of algorithm *SCMP* for computing the product $A_1 A_2 \cdots A_k$. The first it to ignore the sparsity of the matrices and multiply the matrices in $O(n^\omega)$ time. The second is to multiply the matrices, one by one, using the naive algorithm. As the naive algorithm uses at most $O(mn)$ operations when *one* of the matrices contains only m non-zero elements, the total number of operations in this case will be $O((\sum_{i=r}^{k} m_r)n)$. For simplicity, let us consider the case in which each one of the matrices $A_1, \ldots, A_k$ contains m non-zero elements. A simple calculation then shows that *SCMP* is faster then the fast dense matrix multiplication algorithm for

$$m \leq n^{\frac{k-1+\omega}{k}} ,$$

and that it is faster than the naive matrix multiplication algorithm for

$$m \geq \max\{n^{\frac{k-\beta(1+(k-1)\alpha)-1}{(k-1)(1-\beta)}} , n^{1+o(1)}\} .$$

For $k = 2$, these bounds coincide with the bounds obtained in Section 3. For $k = 3$, with the best available bounds on ω, α and β, we get that *SCMP* is the fastest algorithm when $n^{1.24} \leq m \leq n^{1.45}$. For smaller values of m the naive algorithm is the fastest, while for larger values of m the fast dense algorithm is the fastest. Sadly, for $k \geq 4$, with the current values of ω, α and β, the new algorithm never improves on both the naive and the dense algorithms. But, this may change if improved bounds on ω, and especially on α, are obtained.

5 Concluding remarks and open problems

We obtained an improved algorithm for the multiplication of two sparse matrices. The algorithm does not rely on any specific structure of the matrices to be multiplies, just on the fact that they are sparse. The algorithm essentially partitions the matrices to be multiplied into a dense part and a sparse part and uses a fast algebraic algorithm to multiply the dense parts, and the naive algorithm to multiply the sparse parts. We also discussed the possibility of extending the ideas to the product of $k \geq 3$ matrices. For $k = 3$ we obtained some improved results. The new algorithms were presented for square matrices. It is not difficult, however, to extend them to work on rectangular matrices.

The most interesting open problem is whether it is possible to speed up the running time of other operations on sparse matrices. In particular, is it possible to compute the transitive closure of a directed graph on n vertices with m edges in $o(mn)$ time? In particular, is there an $O(m^{1-\epsilon}n^{1+\epsilon})$ algorithm for the problem, for some $\epsilon > 0$?

References

[AYZ95] N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM*, 42:844–856, 1995.

[AYZ97] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17:209–223, 1997.

[BCS97] P. Bürgisser, M. Clausen, and M.A. Shokrollahi. *Algebraic complexity theory*. Springer-Verlag, 1997.

[Cha02] T. Chan. Dynamic subgraph connectivity with geometric applications. In *Proc. of 34th STOC*, pages 7–13, 2002.

[Che97] J. Cheriyan. Randomized $\tilde{O}(M(|V|))$ algorithms for problems in matching theory. *SIAM Journal on Computing*, 26:1635–1655, 1997.

[Chi78] F.Y. Chin. An $O(n)$ algorithm for determining a near-optimal computation order of matrix chain products. *Communications of the ACM*, 21(7):544–549, 1978.

[CLRS01] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms*. The MIT Press, second edition, 2001.

[Coh99] E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM Journal on Computing*, 28:210–236, 1999.

[Cop97] D. Coppersmith. Rectangular matrix multiplication revisited. *Journal of Complexity*, 13:42–49, 1997.

[CU03] H. Cohn and C. Umans. A group-theoretic approach to fast matrix multiplication. In *Proc. of 44th FOCS*, pages 438–449, 2003.

[CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[DI00] C. Demetrescu and G.F. Italiano. Fully dynamic transitive closure: Breaking through the $O(n^2)$ barrier. In *Proceedings of FOCS'00*, pages 381–389, 2000.

[EG03] F. Eisenbrand and F. Grandoni. Detecting directed 4-cycles still faster. *Information Processing Letters*, 87(1):13–15, 2003.

[Gus78] F.G. Gustavson. Two fast algorithms for sparse matrices: Multiplication and permuted transposition. *ACM Transactions on Mathematical Software*, 4(3):250–269, 1978.

[HP98] X. Huang and V.Y. Pan. Fast rectangular matrix multiplications and applications. *Journal of Complexity*, 14:257–299, 1998.

[HS82] T.C. Hu and M.T. Shing. Computation of matrix chain products I. *SIAM Journal on Computing*, 11(2):362–373, 1982.

[HS84] T.C. Hu and M.T. Shing. Computation of matrix chain products II. *SIAM Journal on Computing*, 13(2):228–251, 1984.

[KS03] D. Kratsch and J. Spinrad. Between $O(nm)$ and $O(n^\alpha)$. In *Proc. of 14th SODA*, pages 709–716, 2003.

[MVV87] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987.

[NP85] J. Nešetřil and S. Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

[Pan85] V. Pan. *How to multiply matrices faster*. Lecture notes in computer science, volume 179. Springer-Verlag, 1985.

[Raz03] R. Raz. On the complexity of matrix product. *SIAM Journal on Computing*, 32:1356–1369, 2003.

[RV89] M.O. Rabin and V.V. Vazirani. Maximum matchings in general graphs through randomization. *Journal of Algorithms*, 10:557–567, 1989.

[RZ02] L. Roditty and U. Zwick. Improved dynamic reachability algorithms for directed graphs. In *Proceedings of FOCS'02*, pages 679–689, 2002.

[Sei95] R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51:400–403, 1995.

[Shp03] A. Shpilka. Lower bounds for matrix product. *SIAM Journal on Computing*, 32:1185–1200, 2003.

[Str69] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[SZ99] A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proc. of 40th FOCS*, pages 605–614, 1999.

[YZ04] R. Yuster and U. Zwick. Detecting short directed cycles using rectangular matrix multiplication and dynamic programming. In *Proc. of 15th SODA*, pages 247–253, 2004.

[Zwi02] U. Zwick. All-pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49:289–317, 2002.